



## Consumer Scanner Products 32-Bit DLL for Windows 95, 98, and NT

*About This Manual*

*Table of Contents*

*Chapter 1*

*Chapter 2*

*Index*



# User Guide

72E-39354-01

Revision A

September 1999



***Document Title Variable***  
***Manual***

*Document Part Number*  
*Revision A*  
*September 1999*



© **1998** by Symbol Technologies, Inc. All rights reserved.

No part of this publication may be reproduced or used in any form, or by any electrical or mechanical means, without permission in writing from Symbol. This includes electronic or mechanical means, such as photocopying, recording, or information storage and retrieval systems. The material in this manual is subject to change without notice.

The software is provided strictly on an “as is” basis. All software, including firmware, furnished to the user is on a licensed basis. Symbol grants to the user a non-transferable and non-exclusive license to use each software or firmware program delivered hereunder (licensed program). Except as noted below, such license may not be assigned, sublicensed, or otherwise transferred by the user without prior written consent of Symbol. No right to copy a licensed program in whole or in part is granted, except as permitted under copyright law. The user shall not modify, merge, or incorporate any form or portion of a licensed program with other program material, create a derivative work from a licensed program, or use a licensed program in a network without written permission from Symbol. The user agrees to maintain Symbol’s copyright notice on the licensed programs delivered hereunder, and to include the same on any authorized copies it makes, in whole or in part. The user agrees not to decompile, disassemble, decode, or reverse engineer any licensed program delivered to the user or any portion thereof.

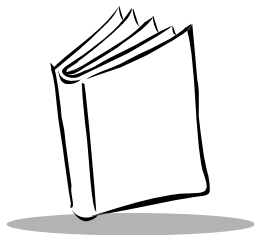
Symbol reserves the right to make changes to any software or product to improve reliability, function, or design.

Symbol does not assume any product liability arising out of, or in connection with, the application or use of any product, circuit, or application described herein.

No license is granted, either expressly or by implication, estoppel, or otherwise under any Symbol Technologies, Inc., intellectual property rights. An implied license only exists for equipment, circuits, and subsystems contained in Symbol products.

Symbol, Spectrum One, and Spectrum24 are registered trademarks of Symbol Technologies, Inc. Other product names mentioned in this manual may be trademarks or registered trademarks of their respective companies and are hereby acknowledged.

Symbol Technologies, Inc.  
One Symbol Plaza  
Holtsville, New York 11742-1300  
<http://www.symbol.com>



# *Contents*

## About This Guide

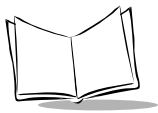
Introduction . . . . .	vii
DLL Improvements . . . . .	vii
Debug Enhancements . . . . .	viii
Notational Conventions . . . . .	x
Service Information . . . . .	x
Symbol Support Centers . . . . .	xi
Warranty . . . . .	xiii
Warranty Coverage and Procedure . . . . .	xiv
General . . . . .	xiv

## Chapter 1. Installing the CSP32.DLL

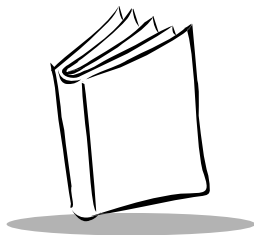
Introduction . . . . .	1-1
Installing CSP32.DLL . . . . .	1-2

## Chapter 2. Function Definitions

Introduction . . . . .	2-1
Returned Status Definitions . . . . .	2-1
Communications . . . . .	2-3
Initialize Communications Port . . . . .	2-3
Restore Communications Port . . . . .	2-5
Basic Function Commands . . . . .	2-6
Read Data . . . . .	2-6
Clear Barcodes . . . . .	2-8
Power Down . . . . .	2-9
CSP Data Get . . . . .	2-10
Get Barcode . . . . .	2-10
Get Device ID . . . . .	2-12
Get Protocol . . . . .	2-14



Get System Status . . . . .	2-16
Get User ID . . . . .	2-18
Get Software Version . . . . .	2-20
CSP Configuration Set . . . . .	2-22
Set TL Bits . . . . .	2-22
Set Volume . . . . .	2-24
Set Barcode Redundancy . . . . .	2-25
Set User ID . . . . .	2-26
Set Continuous Scanning . . . . .	2-28
CSP Configuration Get . . . . .	2-29
Get TL Bits . . . . .	2-29
Get Volume . . . . .	2-31
Get Barcode Redundancy . . . . .	2-32
Get Continuous Scanning . . . . .	2-33
DLL Configuration . . . . .	2-34
Set Retry Count . . . . .	2-34
Get Retry Count . . . . .	2-35
Miscellaneous . . . . .	2-36
Get DLL Version . . . . .	2-36
Debug . . . . .	2-37
Set Debug Mode . . . . .	2-37
Get Communications Port Information . . . . .	2-38
Advanced Function Commands . . . . .	2-40
Read Raw Data . . . . .	2-40
Set Device Parameters . . . . .	2-42
Get Device Parameters . . . . .	2-44
Interrogate the Device . . . . .	2-47



## About This Guide

### Introduction

---

The *Consumer Scanning Products 32-Bit Dynamic Link Library for Windows 95, 98, and Windows NT User Guide* provides the Application Programming Interface (API) definition for the Consumer Scanning Products (CSP) Dynamic Link Library (DLL). Current Symbol offerings in consumer scanning products are the CyberPen and the CS 2000.

Application developers supporting Symbol's Consumer Scanning Products will use the DLL. As a 32-bit DLL, it will function on Windows '95, Windows '98, and Windows NT platforms in applications written in Visual Basic as well as Visual C/C++.

The documentation and demo programs included in this package will provide developers with all of the resources necessary to create full-featured applications.

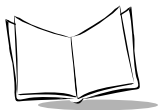
The CSP DLL runs under Windows '95 / '98 and NT 4.0 operating systems. The DLL was created using MS Visual C/C++ V 5.0 compiler. The DLL has been tested with MS Visual C and Visual Basic applications. All of the source code required to modify the demo programs or the DLL are included.

### DLL Improvements

---

The new DLL for the CSP device has been structured so it can be easily used by the Visual 'C' and Visual Basic application programmer.

- ◆ The DLL eliminates the use of structures or pointer to structures. This provides a very simple and straightforward interface to the DLL.
  - ◆ New functions have been added which eliminate the need for complex structures.



- ♦ Simple data types are now used to pass information between the application and the DLL. Data is passed to the DLL as a call by value. Data is retrieved from the DLL as a returned value for single value results or call by reference for character strings.
- ♦ DLL functions hide the complex details of the CSP device operation.
  - ♦ All functions that talk to the CSP device automatically interrogate the device first.
  - ♦ The communications are set to 2400 baud, 8 data bits, and odd parity. The application programmer only has to set the active RS-232 communications port.
- ♦ When compiled in *DEBUG* mode, the DLL provides all the functions of the regular DLL plus debug features. This provides additional help for first time users so that they can get their application debugged and running in a timely manner. These functions include a logging function that writes to a file all commands and responses issued and received from the CSP device and a serial port test. The *DEBUG* version of the DLL is provided to assist users. When its features are no longer required, simply use the *RELEASE* version of the DLL to reduce the size of the DLL and increase its speed.
- ♦ A '.BAS' file is provided with the DLL that defines all the function prototypes and constants that the Visual Basic programmer will need to use the DLL.
- ♦ A '.H' file is provided with the DLL that defines all the function prototypes and constants that the Visual C/C++ programmer will need to use the DLL.
- ♦ The function prototypes are documented for both the Visual Basic and the Visual 'C' user. Two separate documents are provided so that programmers familiar with Visual Basic are not confused by the 'C' prototyping convention and vice versa. Each application programmer can review the function calls and associated documentation in the form with which they are most familiar.
- ♦ Code example snippets in the documentation show a developer how to use the function correctly. The examples will be tailored to match the Visual Basic and Visual 'C' prototypes.

## Debug Enhancements

---

Debug capabilities within the DLL provide the user with the assurance that the DLL is functioning correctly. Two forms of DLL functions are provided specifically for the purpose of testing the DLL with an application.

The first function group allows an application to create a log file of all transactions between the CSP device and the serial port. Commands and responses are written to the log file as they occur when the log file is enabled. Each command/response group is time stamped in the log



file. Each time the log file is enabled, the contents of the previous log file is overwritten. The availability of a log file feature can be a great asset in tracking down protocol issues and verifying CSP device operations when a developer is getting unexpected results. For the release version of the DLL, all log file functions and features are conditionally compiled out to keep the size of the deliverable DLL to a minimum and execution speed reasonable. To minimize any impact on the application developer, the log file functions are stubbed out so that the application code does not need to be changed.

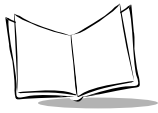
The second function group gives the application developer some much needed information about the serial port availability on the host PC. The user can query the PC about a specific serial port, which might be available for use. The results of the function call tell the application programmer the specified COM port is:

- ◆ an RS-232 device
- ◆ a modem
- ◆ not valid for the current system
- ◆ is currently in use by another program (thus rendering it unusable for the CSP application) or
- ◆ reports some other form of system error related to the device.

Using this information, an application can narrow down the selection of appropriate COM ports for use with the CSP device or provide feedback to a user regarding the availability of the specified port. While listed as part of the debug function group, this function is fully functional in both the debug and release forms of the DLL.

A stand alone debug program is provided to support CSP device application developers and users. The debug program, CommTest.exe, consists of a Windows dialog interface to interact with the developer/user. It provides information about each of the serial ports commonly available on a PC (namely COM1 through COM4). To insure that the CSP device is connected properly, the program can run an “interrogation” test to check for the presence of a CSP device. If the interrogation succeeds, the program updates the display with the current CSP device data obtained during the interrogation.

To resolve any hardware problems that might exist, the debug program is able to test the specified RS-232 serial port through the use of a loop back connector. The loop back connector should connect TX/RX, DTR/DSR, and RTS/CTS signals for testing. An external loop back is required for CSP device port verification because the CyberWell requires the DTR signal to be asserted for communications. The CSP device Protocol does not document this fact, while common knowledge. By using the loop back connector and the debug program, numerous problems can be detected and corrected which are not related to the DLL or the CSP device.



An automated wrap around test verifies the ability of the serial port to send and receive data and verify that the DTR/DSR and RTS/CTS signals are functioning properly.

A function is provided to return the version information about the DLL. The version information for the DLL will be stored in the VERSIONINFO resource statement when the DLL is created. By using the VERSIONINFO resource statement, any Windows program can access the version information associated with a particular DLL or executable. Whenever a new version of the DLL is released, the "File Version" field in the resource file must be manually edited to reflect the correct revision level of the code. The information returned is exactly the information that the user would see on the "Version" tab if they selected the file in the Windows File View and selected Properties.

Since the version information is string based, the returned string can conform to any company's version naming convention. Debug versions of the DLL will automatically prepend "Debug" to the returned version string so that developers can determine which version of the DLL is being used.

## Notational Conventions

---

The following conventions are used in this document:

- ◆ Italics are used to highlight specific items in the general text, and to identify chapters and sections in this and related documents.
- ◆ Bullets (•) indicate:
  - ◆ action items
  - ◆ lists of alternatives
  - ◆ lists of required steps that are not necessarily sequential
- ◆ Sequential lists (e.g., those that describe step-by-step procedures) appear as numbered lists.

## Service Information

---

If you have a problem with your equipment, contact the *Symbol Support Centers*. Before calling, have the model number, serial number, and several of your bar code symbols at hand.

Call the Support Center from a phone near the scanning equipment so that the service person can try to talk you through your problem. If the equipment is found to be working properly and the problem is symbol readability, the Support Center will request samples of your bar codes for analysis at our plant.

If your problem cannot be solved over the phone, you may need to return your equipment for servicing. If that is necessary, you will be given specific directions.

---

**Note:** *Symbol Technologies is not responsible for any damages incurred during shipment if the approved shipping container is not used. Shipping the units improperly can possibly void the warranty. If the original shipping container was not kept, contact Symbol to have another sent to you.*

---

## Symbol Support Centers

For service information, warranty information or technical assistance contact or call the Symbol Support Center in:

### United States

Symbol Technologies, Inc.  
One Symbol Plaza  
Holtsville, New York 11742-1300  
1-800-653-5350

### United Kingdom

Symbol Technologies  
Symbol Place  
Winnersh Triangle, Berkshire RG41 5TP  
United Kingdom  
0800 328 2424 (Inside UK)  
+44 118 945 7529 (Outside UK)

### Australia

Symbol Technologies Pty. Ltd.  
432 St. Kilda Road  
Melbourne, Victoria 3004  
1-800-672-906 (Inside Australia)  
+61-3-9866-6044 (Outside Australia)

### Canada

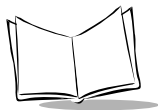
Symbol Technologies Canada, Inc.  
2540 Matheson Boulevard East  
Mississauga, Ontario, Canada L4W 4Z2  
905-629-7226

### Asia/Pacific

Symbol Technologies Asia, Inc.  
230 Victoria Street #04-05  
Bugis Junction Office Tower  
Singapore 188024  
337-6588 (Inside Singapore)  
+65-337-6588 (Outside Singapore)

### Austria

Symbol Technologies Austria GmbH  
Prinz-Eugen Strasse 70  
Suite 3  
2.Haus, 5.Stock  
1040 Vienna, Austria  
1-505-5794 (Inside Austria)  
+43-1-505-5794 (Outside Austria)



### **Denmark**

Symbol Technologies AS  
Gydevang 2,  
DK-3450 Allerød, Denmark  
7020-1718 (Inside Denmark)  
+45-7020-1718 (Outside Denmark)

### **Finland**

Oy Symbol Technologies  
Kaupintie 8 A 6  
FIN-00440 Helsinki, Finland  
9 5407 580 (Inside Finland)  
+358 9 5407 580 (Outside Finland)

### **Germany**

Symbol Technologies GmbH  
Waldstrasse 68  
D-63128 Dietzenbach, Germany  
6074-49020 (Inside Germany)  
+49-6074-49020 (Outside Germany)

### **Latin America Sales Support**

7900 Glades Road  
Suite 340  
Boca Raton, Florida 33434 USA  
1-800-347-0178 (Inside United States)  
+1-561-483-1275 (Outside United States)

### **Netherlands**

Symbol Technologies  
Kerkplein 2, 7051 CX  
Postbus 24 7050 AA  
Varsseveld, Netherlands  
315-271700 (Inside Netherlands)  
+31-315-271700 (Outside Netherlands)

### **Europe/Mid-East Distributor Operations**

Contact your local distributor or call  
+44 118 945 7360

### **France**

Symbol Technologies France  
Centre d'Affaire d'Antony  
3 Rue de la Renaissance  
92184 Antony Cedex, France  
01-40-96-52-21 (Inside France)  
+33-1-40-96-52-50 (Outside France)

### **Italy**

Symbol Technologies Italia S.R.L.  
Via Cristoforo Columbo, 49  
20090 Trezzano S/N Naviglio  
Milano, Italy  
2-484441 (Inside Italy)  
+39-02-484441 (Outside Italy)

### **Mexico**

Symbol Technologies Mexico Ltd.  
Torre Picasso  
Boulevard Manuel Avila Camacho No 88  
Lomas de Chapultepec CP 11000  
Mexico City, DF, Mexico  
5-520-1835 (Inside Mexico)  
+52-5-520-1835 (Outside Mexico)

### **Norway**

Symbol Technologies  
Trollasveien 36  
Postboks 72  
1414 Trollasen, Norway  
66810600 (Inside Norway)  
+47-66810600 (Outside Norway)

### **South Africa**

Symbol Technologies Africa Inc.  
Block B2  
Rutherford Estate  
1 Scott Street  
Waverly 2090 Johannesburg  
Republic of South Africa  
11-4405668 (Inside South Africa)  
+27-11-4405668 (Outside South Africa)

### **Spain**

Symbol Technologies S.A.  
Edificio la Piovera Azul  
C. Peonias, No. 2 - Sexta Planta  
28042 Madrid, Spain  
9-1-320-39-09 (Inside Spain)  
+34-9-1-320-39-09 (Outside Spain)

### **Sweden**

Symbol Technologies AB  
Albygatan 109D  
Solna  
Sweden  
84452900 (Inside Sweden)  
+46 84452900 (Outside Sweden)

If you purchased your Symbol product from a Symbol Business Partner, contact that Business Partner for service.

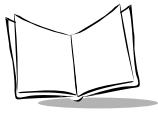
## **Warranty**

---

Symbol Technologies, Inc ("Symbol") manufactures its hardware products in accordance with industry-standard practices. Symbol warrants that for a period of twelve (12) months from date of shipment, products will be free from defects in materials and workmanship.

This warranty is provided to the original owner only and is not transferable to any third party. It shall not apply to any product (i) which has been repaired or altered unless done or approved by Symbol, (ii) which has not been maintained in accordance with any operating or handling instructions supplied by Symbol, (iii) which has been subjected to unusual physical or electrical stress, misuse, abuse, power shortage, negligence or accident or (iv) which has been used other than in accordance with the product operating and handling instructions. Preventive maintenance is the responsibility of customer and is not covered under this warranty.

Wear items and accessories having a Symbol serial number, will carry a 90-day limited warranty. Non-serialized items will carry a 30-day limited warranty.



## **Warranty Coverage and Procedure**

During the warranty period, Symbol will repair or replace defective products returned to Symbol's manufacturing plan in the US. For warranty service in North America, call the Symbol Support Center at 1-800-653-5350. International customers should contact the local Symbol office or support center. If warranty service is required, Symbol will issue a Return Material Authorization Number. Products must be shipped in the original or comparable packaging, shipping and insurance charges prepaid. Symbol will ship the repaired or replacement product freight and insurance prepaid in North America. Shipments from the US or other locations will be made F.O.B. Symbol's manufacturing plant.

Symbol will use new or refurbished parts at its discretion and will own all parts removed from repaired products. Customer will pay for the replacement product in case it does not return the replaced product to Symbol within 3 days of receipt of the replacement product. The process for return and customer's charges will be in accordance with Symbol's Exchange Policy in effect at the time of the exchange.

Customer accepts full responsibility for its software and data including the appropriate backup thereof. Repair or replacement of a product during warranty will not extend the original warranty term.

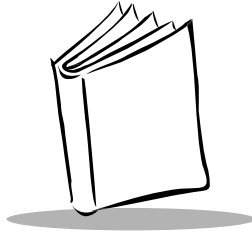
Symbol's Customer Service organization offers an array of service plans, such as on-site, depot, or phone support, that can be implemented to meet customer's special operational requirements and are available at a substantial discount during warranty period.

## **General**

Except for the warranties stated above, Symbol disclaims all warranties, express or implied, on products furnished hereunder, including without limitation implied warranties of merchantability and fitness for a particular purpose. The stated express warranties are in lieu of all obligations or liabilities on part of Symbol for damages, including without limitation, special, indirect, or consequential damages arising out of or in connection with the use or performance of the product.

Seller's liability for damages to buyer or others resulting from the use of any product, shall in no way exceed the purchase price of said product, except in instances of injury to persons or property.

Some states (or jurisdictions) do not allow the exclusion or limitation of incidental or consequential damages, so the proceeding exclusion or limitation may not apply to you.



# Chapter 1

## Installing the CSP32.DLL

### Introduction

---

This chapter covers the installation of the CSP32.DLL on your Windows 95, 98 and NT 4.0 system.

If you will be redistributing the Csp32.dll with your system source code, you need to be aware of other DLLs that are required in order for the Csp32.dll to function properly. The Csp32.dll relies on the files specified in the following table in order to operate. Some of the files are provided by the Windows operating system; others are provided by Visual C/C++ and Visual Basic installations.

---

**Note:** *If you are installing programs on a system which does not have Visual C/C++ or Visual Basic installed, then you are responsible for installing the distributable Windows DLLs into the \Windows\System directory. Failure to do so will cause error messages such as “Error 48: Error in loading DLL Csp32.dll.” Error messages such as this indicate that the Csp32.dll could not be loaded because it relies on other DLLs, which are not present, in order to function. Consult your Visual Basic documentation for a list of other files which are required to execute Visual Basic programs.*

---

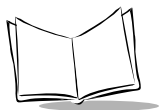


Table 1-1. CSP32.DLL Files

Debug	Release	Windows System Files
✓	✓	\Windows\System\Version.dll
✓	✓	\Windows\System\User32.dll
✓	✓	\Windows\System\Kernel.dll
Debug	Release	Microsoft Distributable Files
	✓	\Windows\System\Mfc42.dll
	✓	\Windows\System\Msvcrt.dll
✓		\Windows\System\Mfc42d.dll
✓		\Windows\System\Msvcrt.d.dll

## Installing CSP32.DLL

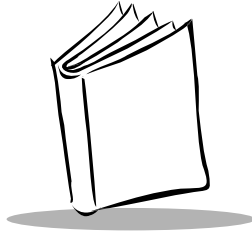
---

To install the CSP32.DLL on your development PC, run the setup.exe program on the CD. Follow the prompts on the screen until you receive a message that the software has been successfully installed. Once the installation completes, you are prompted to reboot your PC in order to use the software.

The CSP setup.exe program adds the distributable files to your \Windows\System directory if they did not already exist. This allows the demo programs that ship with the Csp32.dll to function immediately after installation.

The Csp32.dll should also be installed into the \Windows\System directory WHEN you are redistributing the DLL. For the purpose of testing, the Csp32.dll has been copied from the \Csp32\Debug directory to each of the demo program directories. Windows will search for the file starting in the current directory, then the Windows\System directory, then by traversing the PATH information. The first instance of the Csp32.dll to be located will be used. By copying the DLL to the local directory you can experiment with the debug and non-debug versions of the code.





## *Chapter 2*

# *Function Definitions*

## **Introduction**

---

This section provides all the available function calls and their descriptions. All of the CSP functions listed below are broken down by functional group.

Please note, the returned data from each function, if negative, indicates an error condition in executing the function, except for errors reported from Windows file operations. A zero or positive value indicates success and/or returned data as detailed by each function description.

For the purpose of compatibility between Visual Basic programs and Visual C programs, the “int” data type is not used within the CSP API functions. This is because a Visual Basic program running under Windows '95, '98, or NT uses a 16-bit value when “int” is specified. Visual C/C++ programs running on the same operating systems use a 32-bit value for “int” data types. Both Visual Basic and Visual C/C++ use a 32-bit data type when specifying “long” variables. So, by specifying “long” variables, the functions calls and documentation across development platforms remains consistent.

## **Returned Status Definitions**

---

The functions in this document may return status as indicated. The standard returned status codes are as follows:

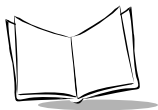


Table 2-1. Status Codes

Return Code	Meaning
<b>STATUS_OK</b>	the function parameters were verified, the function call was successful.
<b>COMMUNICATIONS_ERROR</b>	indicates that a timeout condition occurred. The previous transmit message was not verified through an acknowledge, and therefore, is questionable.
<b>BAD_PARAM</b>	indicates that the communications parameters used were out of limits for the serial communications setup.
<b>SETUP_ERROR</b>	indicates that a parameter passed to a function is out of limits.
<b>INVALID_COMMAND_NUMBER</b>	returned by the CSP Device, indicates that the command was not received correctly.
<b>NO_ERROR_ENCOUNTED</b>	returned by the CSP Device to indicate the receipt of a valid command sequence.
<b>COMMAND_LRC_ERROR</b>	returned by the CSP Device to indicate that a command packet was invalid due to the failure of the Longitudinal Redundancy Check.
<b>RECEIVED_CHARACTER_ERROR</b>	returned by the CSP Device to indicate the failure of a character within a packet.
<b>GENERAL_ERROR</b>	either a catchall error returned by the CSP Device, or an error reporting that a NULL pointer was sent to a function when requesting CSP Device parameter settings.
<b>ACCESS_DENIED</b>	the specified communications port is currently in use by another program (for example HyperTerminal) on the host computer.
<b>FILE_NOT_FOUND</b>	the specified communications port does not exist on the host system.

# Communications

---

## *Initialize Communications Port*

### Syntax

```
long cspInit(long nComPort )
```

### Description

The **cspInit()** function provides a single call to initialize the CSP communications and all of the software structures for the CSP interface. The initialization will consist of:

- ♦ Initialize all memory structures, queues, and pointers
- ♦ Establish a serial communications connection with the CSP using the specified port.

If any errors are detected, the serial communications will be unchanged.

### Parameters

Where:

*nComPort* is one of the following values:

```
COM1
COM2
COM3
COM4
```

### Returned Status

```
STATUS_OK
BAD_PARAM if nComPort is less than COM1
COMMUNICATIONS_ERROR
```

### Example

```
// try to initialize COM1...
If (cspInit(COM1) == STATUS_OK)
{
    // we can use COM1 to talk to a CSP device...
}
else
{
```



## *32-Bit DLL for Windows 95, 98, and Windows NT User Guide*

```
// COM1 is not available  
}
```

## Restore Communications Port

### Syntax

```
long cspRestore( void )
```

### Description

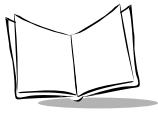
The **cspRestore()** function provides a single call to restore the previously selected COM port and all of the software structures for the CSP interface.

### Returned Status

STATUS\_OK  
COMMUNICATIONS\_ERROR

### Example

```
// try to release the COM port...
If (cspRestore() == STATUS_OK)
{
    // released the COM port, switch to a new port/new device...
}
else
{
    // error handling procedure
}
```



## Basic Function Commands

---

### *Read Data*

#### Syntax

```
long cspReadData( void )
```

#### Description

The **cspReadData()** function reads both the Barcode and the Device ID information stored in the CSP device and saves this data in the DLL. The user must then call other functions listed below to retrieve this data. Please refer to the following functions to retrieve the data from the DLL.

- ♦ **cspGetBarcode()** on page 2-10
- ♦ **cspGetDeviceID()** on page 2-12

#### Returned Status

If the number is zero the operation was successful, but no barcodes were stored in the CSP. The Device ID can be read using **cspGetDeviceID()**.

If the number is greater than zero: Indicates the number of barcodes retrieved from the CSP. These barcodes can be read using **cspGetBarcode()**.

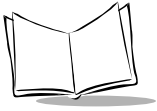
Or if value was negative:

```
COMMUNICATIONS_ERROR  
INVALID_COMMAND_NUMBER  
COMMAND_LRC_ERROR  
RECEIVED_CHARACTER_ERROR  
GENERAL_ERROR
```

#### Example

```
long nNumBarcodes;  
  
// initialize the COM port for its FIRST use...  
If (cspInit(COM1) == STATUS_OK)  
{  
    // read in the barcodes...
```

```
if ((nNumBarcodes = cspReadData()) >= 0)
    {
        // nNumBarcodes indicates how many barcodes were
        // stored
    }
else
// Error: alert the user to activate the CSP device
    }
```



## Clear Barcodes

### Syntax

```
long cspClearBarCodes( void )
```

### Description

The **cspClearBarCodes()** function clears all the barcodes stored in the CSP.

If this is the last action in a communications session, it should be followed by a power down command.

### Returned Status

STATUS\_OK  
COMMUNICATIONS\_ERROR  
INVALID\_COMMAND\_NUMBER  
COMMAND\_LRC\_ERROR  
RECEIVED\_CHARACTER\_ERROR  
GENERAL\_ERROR

### Example

```
// try to release the COM port...
If (cspClearBarCodes () == STATUS_OK)
{
    // cleared the old barcodes from the device
}
else
{
    // Error: alert the user to activate the CSP device
}
```



## Power Down

### Syntax

```
long cspPowerDown( void )
```

### Description

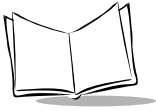
The **cspPowerDown()** function directs the device to end the communication session and enter the STOP mode until the next trigger action. This function returns on or before the device issues an audible signal confirming the power down command.

### Returned Status

```
STATUS_OK
COMMUNICATIONS_ERROR
INVALID_COMMAND_NUMBER
COMMAND_LRC_ERROR
RECEIVED_CHARACTER_ERROR
GENERAL_ERROR
```

### Example

```
long nNumBarcodes;
// initialize the COM port for its FIRST use...
If (cspInit(COM1) == STATUS_OK)
{
    // read in the barcodes...
    if ((nNumBarcodes = cspReadData()) >= 0)
    {
        // nNumBarcodes indicates how many barcodes were
        // stored
        // Because the barcodes are buffered by the DLL,
        // conserve battery life and power down the device
        cspPowerDown();
    }
    else
        // Error: alert the user to activate the CSP device
}
```



## CSP Data Get

---

### Get Barcode

#### Syntax

```
long cspGetBarcode (          char szBarData[ ],  
                        long nBarcodeNumber,  
                        long nMaxLength )  
/
```

#### Description

The **cspGetBarcode()** function copies the barcode data to the users allocated memory space from the DLL. The function will only copy up to the *nMaxLength* characters. If the barcode is longer it will be truncated. The barcode data will be null terminated. The user must specify which barcode they want (refer to **cspReadData()** on page 2-6) and the maximum length of the allocated space they have reserved.

#### Parameters

Where:

*szBarData* is a character array the user has allocated to hold the barcodedata. The string will be null terminated.

*nBarcodeNumber* is the number of the barcode the user wants to read.

*nMaxLength* is the length of the allocated space including the null terminator. If *nMaxLength* is set to DETERMINE\_SIZE, the function will return the length of the barcode without copying any data. Note, if the user specifies the space is bigger than was actually allocated, unpredictable results will occur.

#### Returned Status

length of barcode

BAD\_PARAM if the user requested a barcode that does not exist.

---

**Note:** *The value returned by this function will reflect the value of the parameter in the CSP device at the time of the last call to the **cspReadData()** function. In order to receive an accurate current*

*parameter value, be sure that the call to `cspReadData()` function is current.*

---

## Example

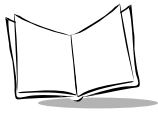
```

long nNumBarcodes;
char szBarData[80];
long nBarcodeNumber;
// initialize the COM port for its FIRST use...
If (cspInit(COM1) == STATUS_OK)
{
    // read in the barcodes...
    if ((nNumBarcodes = cspReadData()) >= 0)
    {
        // nNumBarcodes indicates how many barcodes were
        // stored
        // Because the barcodes are buffered by the DLL,
        // conserve battery life and power down the CSP device
        cspPowerDown();

        // read each of the barcodes from the device...
        for ( nBarcodeNumber = 0;
              nBarcodeNumber < nNumBarcodes;
              nBarcodeNumber++)
        {
            cspGetBarcode ( szBarData,
                           nBarcodeNumber,
                           sizeof(szBarData) );

            // store the null terminated szBarData[] string
            // to a disk file...
        }
    }
else
    // Error: alert the user to activate the CSP device
}

```



## Get Device ID

### Syntax

```
long cspGetDeviceId( char szDeviceId[9], long nMaxLength )
```

### Description

The **cspGetDeviceID()** function retrieves the *szDeviceId* [] string from the CSP device data stored in the DLL. The string is null terminated.

### Parameters

Where:

*szDeviceId* [] holds the returned data.

*nMaxLength* is the length of the allocated space including the null terminator. Note, if the user specifies the space is bigger than was actually allocated, unpredictable results will occur.

### Returned Status

The length of *szDeviceId*.

---

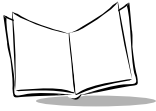
**Note:** *The value returned by this function will reflect the value of the parameter in the device at the time of the last call to the **cspReadData()** function. In order to receive an accurate current parameter value, be sure that the call to **cspReadData()** function is current.*

---

### Example

```
long nNumBarcodes;
char szDeviceId[9];
// initialize the COM port for its FIRST use...
If (cspInit(COM1) == STATUS_OK)
{
    // read in the barcodes...
    if ((nNumBarcodes = cspReadData()) >= 0)
    {
        // nNumBarcodes indicates how many barcodes were
        // stored
```

```
// Because the barcodes are buffered by the DLL,  
// conserve battery life and power down the device  
cspPowerDown();  
  
// retrieve the null terminated szDeviceId[] string and  
// write it to a file..  
cspGetDeviceId(szDeviceId, sizeof(szDeviceId));  
}  
else  
    // Error: alert the user to activate the CSP device  
}
```



## Get Protocol

### Syntax

```
long cspGetProtocol( void )
```

### Description

The **cspGetProtocol()** returns a long word corresponding to the protocol byte from the CSP device data stored in the DLL.

### Returned Status

If zero or positive, Protocol version

If negative, the Protocol version for the last transfer is not available.

---

**Note:** *The data returned by this function will reflect the value of the parameter in the CSP device at the time of the last call to one of the following functions:*

---

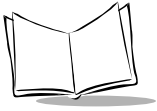
```
cspReadData(),  
cspSet...(),  
cspGet...(),  
cspClearBarcode(),  
cspPowerDown()
```

In order to receive an accurate current parameter value, be sure that the call to these functions is current.

### Example

```
long nNumBarcodes;  
long nProtocol;  
  
// initialize the COM port for its FIRST use...  
If (cspInit(COM1) == STATUS_OK)  
{  
    // read in the barcodes...  
    if ((nNumBarcodes = cspReadData()) >= 0)  
    {  
        // nNumBarcodes indicates how many barcodes were  
        // stored
```

```
// Because the barcodes are buffered by the DLL,  
// conserve battery life and power down the device  
cspPowerDown();  
  
// retrieve the CSP protocol value...  
nProtocol = cspGetProtocol ();  
}  
else  
    // Error: alert the user to activate the CSP device  
}
```



## Get System Status

### Syntax

```
long cspGetSystemStatus( void )
```

### Description

The **cspGetSystemStatus()** returns a long word corresponding to the system status byte from the CSP device data stored in the DLL.

### Returned Status

If zero or positive, system status

If negative, the system status for the last transfer is not available.

---

**Note:** *The data returned by this function will reflect the value of the parameter in the device at the time of the last call to one of the following functions:*

---

```
cspReadBarCodes(),  
cspSet...(),  
cspGet...(),  
cspClearBarcode(),  
cspPowerDown()
```

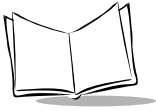
In order to receive an accurate current parameter value, be sure that the call to these functions is current.

### Example

```
long nNumBarcodes;  
long nStatus;  
  
// initialize the COM port for its FIRST use...  
If (cspInit(COM1) == STATUS_OK)  
{  
    // read in the barcodes...  
    if ((nNumBarcodes = cspReadData()) >= 0)  
    {  
        // nNumBarcodes indicates how many barcodes were  
        // stored
```



```
// Because the barcodes are buffered by the DLL,  
// conserve battery life and power down the device  
cspPowerDown();  
  
// retrieve the CSP status value...  
nStatus = cspGetSystemStatus ();  
}  
else  
    // Error: alert the user to activate the CSP device  
}
```



## Get User ID

### Syntax

```
long cspGetUserID( char szUserId[9], long nMaxLength )
```

### Description

The **cspGetUserID** function retrieves the *szUserId* [] string from the CSP device data stored in the DLL. The returned string is null terminated.

Where:

*szUserId* [] holds the returned data.

*nMaxLength* is the length of the allocated space including the null terminator. Note, if the user specifies the space is bigger than was actually allocated, unpredictable results will occur.

### Returned Status

The length of *szUserId*.

COMMUNICATIONS\_ERROR  
INVALID\_COMMAND\_NUMBER  
COMMAND\_LRC\_ERROR  
RECEIVED\_CHARACTER\_ERROR  
GENERAL\_ERROR

---

**Note:** *The data returned by this function will reflect the value of the parameter in the device at the time of the last call to one of the following function:*

---

cspReadBarCodes(),  
cspSet...(),  
cspGet...(),  
cspClearBarcode(),  
cspPowerDown()

In order to receive an accurate current parameter value, be sure that the call to these functions is current.

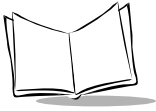
### Example

```
long nNumBarcodes;
char szUserId[9];

// initialize the COM port for its FIRST use...
If (cspInit(COM1) == STATUS_OK)
{
    // read in the barcodes...
    if ((nNumBarcodes = cspReadData()) >= 0)
    {
        // nNumBarcodes indicates how many barcodes were
        // stored

        // Because the barcodes are buffered by the DLL,
        // conserve battery life and power down the device
        cspPowerDown();

        // retrieve the null terminated szUserId[] string...
        cspGetUserID(szUserId, sizeof(szUserId));
    }
else
    // Error: alert the user to activate the CSP device
}
```



## Get Software Version

### Syntax

```
long cspGetSwVersion( char szSwVersion[9], long nMaxLength )
```

### Description

The **cspGetSWVersion()** function retrieves the user *szSwVersion* [] string from the CSP device data stored in the DLL.

Where:

*szSwVersion* [] holds the null terminated version string.

*nMaxLength* is the length of the allocated space including the null terminator. Note, if the user specifies the space is bigger than was actually allocated, unpredictable results will occur.

### Returned Status

The length of *szSwVersion*.

---

**Note:** *The data returned by this function will reflect the value of the parameter in the device at the time of the last call to one of the following functions:*

---

```
cspReadBarCodes(),  
cspSet...(),  
cspGet...(),  
cspClearBarcode(),  
cspPowerDown()
```

In order to receive an accurate current parameter value, be sure that the call to these functions is current.

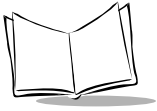
### Example

```
long nNumBarcodes;  
char szSwVersion [9];  
  
// initialize the COM port for its FIRST use...  
If (cspInit(COM1) == STATUS_OK)  
{
```

```
// read in the barcodes...
if ((nNumBarcodes = cspReadData()) >= 0)
{
    // nNumBarcodes indicates how many barcodes were
    // stored

    // Because the barcodes are buffered by the DLL,
    // conserve battery life and power down the device
    cspPowerDown();

    // retrieve the null terminated szSwVersion [] string...
    cspGetSwVersion(szSwVersion, sizeof(szSwVersion));
}
else
    // Error: alert the user to activate the CSP device}
```



## CSP Configuration Set

---

### Set TL Bits

#### Syntax

```
long cspSetTlBits( char aTlBits[8], long nMaxLength )
```

#### Description

The **cspSetTLBits()** function saves 8 bytes (64 bits) of host specific data to the CSP device.

#### Parameter

Where:

*aTlBits [8]* specifies all of the bits which are to be stored.

*nMaxLength* is the length of the allocated. Note, if the user specifies the space is bigger than was actually allocated, unpredictable results will occur.

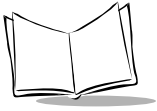
#### Returned Status

STATUS\_OK  
BAD\_PARAM if *nMaxLength* < 8  
COMMUNICATIONS\_ERROR  
INVALID\_COMMAND\_NUMBER  
COMMAND\_LRC\_ERROR  
RECEIVED\_CHARACTER\_ERROR  
GENERAL\_ERROR

#### Example

```
char count;  
char aTlBits[8];  
  
// initialize the array of bits...  
for (count = 0; count < sizeof(aTlBits); count++)  
    aTlBits[count] = count;  
  
// initialize the COM port for its FIRST use...  
If (cspInit(COM1) == STATUS_OK)  
{  
    // program the new array of aTlBits[...]
```

```
        cspSetTlBits(aTlBits, sizeof(aTlBits));  
    }  
else  
    {  
        // Error: alert the user to activate the CSP device  
    }
```



## Set Volume

### Syntax

```
long cspSetVolume(long nVolume )
```

### Description

The **cspSetVolume()** function sets the CSP device's volume.

### Parameter

Where:

*nVolume* is one of the following values:

VOLUME\_QUIET  
VOLUME\_LOW  
VOLUME\_MEDIUM  
VOLUME\_HIGH

### Returned Status

STATUS\_OK  
COMMUNICATIONS\_ERROR  
INVALID\_COMMAND\_NUMBER  
COMMAND\_LRC\_ERROR  
RECEIVED\_CHARACTER\_ERROR  
GENERAL\_ERROR  
BAD\_PARM if *nVolume* not in range of valid values

### Example

```
// initialize the COM port for its FIRST use...
If (cspInit(COM1) == STATUS_OK)
{
    // read in the barcodes...
    if (cspSetVolume (VOLUME_MEDIUM) == STATUS_OK)
    {
        // new volume was set properly...
    }
    else
        // Error: alert the user to activate the CSP device
}
```



## Set Barcode Redundancy

### Syntax

```
long cspSetBarcodeRedundancy(long nOnOff )
```

### Description

The **cspSetBarcodeRedundancy()** function turns the barcode redundancy on or off.

### Parameter

Where:

*nOnOff* is one of the following values:

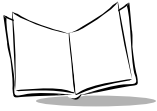
```
    PARM_OFF  
    PARM_ON
```

### Returned Status

```
STATUS_OK  
COMMUNICATIONS_ERROR  
INVALID_COMMAND_NUMBER  
COMMAND_LRC_ERROR  
RECEIVED_CHARACTER_ERROR  
GENERAL_ERROR  
BAD_PARM if nOnOff not in range of valid values
```

### Example

```
// initialize the COM port for its FIRST use...  
If (cspInit(COM1) == STATUS_OK)  
{  
    // read in the barcodes...  
    if (cspSetBarcodeRedundancy (PARM_OFF) == STATUS_OK)  
    {  
        // Barcode Redundancy was disabled...  
    }  
    else  
        // Error: alert the user to activate the CSP device  
}
```



## Set User ID

### Syntax

```
long cspSetUserId( char szUserId[9], long nMaxLength )
```

### Description

The **cspSetUserID()** function saves the *szUserId []* string to the CSP device.

Where:

*szUserId []* contains the string to be written to the CSP device.

*nMaxLength* is the length of the allocated space including the null terminator. Note, if the user specifies the space is bigger than was actually allocated, unpredictable results will occur.

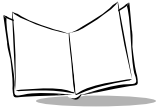
### Returned Status

STATUS\_OK  
COMMUNICATIONS\_ERROR  
INVALID\_COMMAND\_NUMBER  
COMMAND\_LRC\_ERROR  
RECEIVED\_CHARACTER\_ERROR  
GENERAL\_ERROR  
BAD\_PARAM if *nMaxLength* is less than 8

### Example

```
char count;  
char szUserId [9];  
  
// initialize the null terminated szUserId[] string..  
memcpy(szUserId, "Its Mine", 9);  
  
// initialize the COM port for its FIRST use..  
If (cspInit(COM1) == STATUS_OK)  
{  
    // program the szUserId []..  
    cspSetUserId (szUserId, sizeof(szUserId));  
}  
else  
{
```

```
// Error: alert the user to activate the CSP device  
}
```



## Set Continuous Scanning

### Syntax

```
long cspSetContinuousScanning(long nOnOff )
```

### Description

The **cspSetContinuousScanning()** function turns the barcode continuous scanning on or off.

### Parameters

Where:

*nOnOff* is one of the following values:

PARM\_OFF  
PARM\_ON

### Returned Status:

STATUS\_OK  
COMMUNICATIONS\_ERROR  
INVALID\_COMMAND\_NUMBER  
COMMAND\_LRC\_ERROR  
RECEIVED\_CHARACTER\_ERROR  
GENERAL\_ERROR  
BAD\_PARM if *nOnOff* not in range of valid values

### Example

```
// initialize the COM port for its FIRST use...
If (cspInit(COM1) == STATUS_OK)
{
    // read in the barcodes...
    if (cspSetContinuousScanning (PARM_OFF) == STATUS_OK)
    {
        // Continuous Scanning was disabled...
    }
    else
        // Error: alert the user to activate the CSP device
}
```

# CSP Configuration Get

---

## Get TL Bits

### Syntax

```
long cspGetTlBits( char aTlBits[8], long nMaxLength )
```

### Description

The **cspGetTLBits()** function retrieves the *aTlBits []* string from the CSP device.

### Parameters

Where:

*aTlBits []* holds the returned data.

*nMaxLength* is the length of the allocated space. Note, if the user specifies the space is bigger than was actually allocated, unpredictable results will occur.

### Returned Status

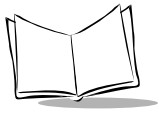
If operation OK- length of *aTlBits* read

```
COMMUNICATIONS_ERROR
INVALID_COMMAND_NUMBER
COMMAND_LRC_ERROR
RECEIVED_CHARACTER_ERROR
GENERAL_ERROR
```

### Example

```
char aTlBits[8];

// initialize the COM port for its FIRST use...
If (cspInit(COM1) == STATUS_OK)
{
    if (cspGetTlBits (aTlBits, sizeof(aTlBits) >= STATUS_OK)
        {
            // retrieved the current value of aTlBits[]
            // from the device...
        }
}
else
{
```



## *32-Bit DLL for Windows 95, 98, and Windows NT User Guide*

```
// Error: alert the user to activate the CSP device  
}  
  
}
```

## Get Volume

### Syntax

```
long cspGetVolume( void )
```

### Description

The **cspGetVolume()** function retrieves the CSP device's volume setting.

### Returned Status

If the returned value is zero or greater:

```
VOLUME_QUIET
VOLUME_LOW
VOLUME_MEDIUM
VOLUME_HIGH
```

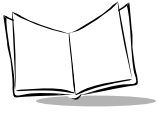
If the returned value is negative:

```
COMMUNICATIONS_ERROR
INVALID_COMMAND_NUMBER
COMMAND_LRC_ERROR
RECEIVED_CHARACTER_ERROR
GENERAL_ERROR
```

### Example

```
long    nVolume;

// initialize the COM port for its FIRST use...
If (cspInit(COM1) == STATUS_OK)
{
    if ((nVolume = cspGetVolume()) >= 0)
    {
        // retrieved the current value of volume from
        // the device...
    }
    else
    {
        // Error: alert the user to activate the CSP device
    }
}
```



## Get Barcode Redundancy

### Syntax

```
long cspGetBarcodeRedundancy( void )
```

### Description

The **cspGetBarcodeRedundancy()** function retrieves the CSP device's redundancy setting.

### Returned Status

If the returned value is zero or greater:

PARM\_OFF  
PARM\_ON

If the returned value is negative:

COMMUNICATIONS\_ERROR  
INVALID\_COMMAND\_NUMBER  
COMMAND\_LRC\_ERROR  
RECEIVED\_CHARACTER\_ERROR  
GENERAL\_ERROR

### Example

```
long    nBcRedundancy;  
// initialize the COM port for its FIRST use...  
If (cspInit(COM1) == STATUS_OK)  
{  
    if ((nBcRedundancy = cspGetBarcodeRedundancy ()) >= 0)  
    {  
        // retrieved the current value of BC Redundancy  
        // from the device...  
    }  
    else  
    {  
        // Error: alert the user to activate the CSP device  
    }  
}
```



## Get Continuous Scanning

### Syntax

```
long cspGetContinuousScanning( void )
```

### Description

The **cspGetContinuousScanning()** function retrieves the CSP device's continuous scanning setting.

### Returned Status

If the returned value is zero or greater:

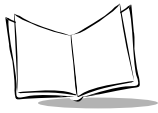
```
PARM_OFF
PARM_ON
```

If the returned value is negative:

```
COMMUNICATIONS_ERROR
INVALID_COMMAND_NUMBER
COMMAND_LRC_ERROR
RECEIVED_CHARACTER_ERROR
GENERAL_ERROR
```

### Example:

```
long    nContinuousScan;
// initialize the COM port for its FIRST use...
If (cspInit(COM1) == STATUS_OK)
{
    if ((nContinuousScan = cspGetContinuousScanning ()) >= 0)
    {
        // retrieved the current value of Continuous Scan
        // from the device...
    }
    else
    {
        // Error: alert the user to activate the CSP device
    }
}
```



## DLL Configuration

---

### Set Retry Count

#### Syntax

```
long cspSetRetryCount(long nRetryCount )
```

#### Description

The **cspSetRetryCount()** function sets the communications retry count. The default is 5. In the event of a communications failure, the DLL will attempt the communication *nRetryCount* more times before returning a communications error. Each retry extends the length of the CSP communication by approximately 1 second until the count expires or the CSP device responds.

#### Parameters

Where:

*nRetryCount* - any long word in the range from 0 to 9.

#### Returned Status

STATUS\_OK  
BAD\_PARAM

#### Example

```
// change the default retry count to 0
// future attempts to get/set device parameters will not attempt
// any retries so the device will time out within one second
cspSetRetryCount(0);
```

## **Get Retry Count**

### **Syntax**

```
long cspGetRetryCount( void )
```

### **Description**

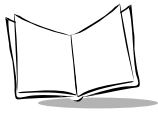
The **cspGetRetryCount()** function retrieves the current retry count setting in the DLL.

### **Returned Status**

The value of the retry count (0-9)

### **Example**

```
long nRetryCount;  
// retrieve the default number of retries upon  
// starting an application  
nRetryCount = cspGetRetryCount ();
```



## Miscellaneous

---

### Get DLL Version

#### Syntax

```
long cspGetDllVersion( char szDllVersion[], long nMaxLength )
```

#### Description

The **cspGetDLLVersion()** function copies the DLL version string into the users allocated memory area. The string is null terminated. The user should allocate a minimum string array size of 20 characters for this function. If the debug version of the DLL is being used, the version information will be prepended with “Debug”.

#### Parameters

Where:

*szDllVersion []* holds the returned data.

*nMaxLength* is the length of the allocated space including the null terminator. If *nMaxLength* is set less than the length of the version string, the function will return the length of the *szDllVersion* without copying any data. Note, if the user specifies the space is bigger than was actually allocated, unpredictable results will occur.

#### Returned Status

STATUS\_OK  
FILE\_NOT\_FOUND  
Length of version string

#### Example

```
char szDllVersion[20];           // minimum of 20 chars recommended
// retrieve the current null terminated DLL
// revision information string...
cspGetDllVersion(szDllVersion, sizeof(szDllVersion));
```

# Debug

---

---

**Note:** *This function only works in the debug version of the DLL.*

---

## Set Debug Mode

### Syntax

```
long cspSetDebugMode(long nOnOff )
```

### Description

The **cspSetDebugMode()** function turns the debug mode of the DLL on or off. Note this function is only writes data to the output file in the debug version of the DLL.

When the debug mode is on, all commands and responses are written to a debug file named 'debug.txt'. Each entry in the file will have a time stamp preceding the entry.

### Parameters

Where:

*nOnOff* is one of the following values:

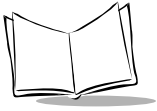
PARM\_OFF  
PARM\_ON

### Returned Status

STATUS\_OK  
FILE\_NOT\_FOUND  
Other: see WinError.h

### Example

```
If (cspSetDebugMode(PARM_ON) != STATUS_OK)
{
    // File error, look up the problem...
}
else
{
    // the debug.txt file will be written to during CSP
    // transfers if the debug version of the DLL is being used...
}
```



## Get Communications Port Information

### Syntax

```
long cspGetCommInfo(long nComPort)
```

### Description

The `cspGetCommInfo()` function determines information about the specified *nComPort*.

This function determines if the Port specified is a valid RS-232 serial port. It can detect the following information about the specified serial port:

- ◆ It can detect a serial port (default RS-232)
- ◆ It can detect a modem (internal/external)
- ◆ It can detect when a port is in use
- ◆ It can detect that the port does not exist
- ◆ It can report other Windows errors associated with the specified port

### Parameters

Where:

*nComPort* is one of the following values:

COM1  
COM2  
COM3  
COM4

### Returned Status

SERIAL\_MODEM - the port is a modem/connected to a modem

SERIAL\_RS232 - the port is an RS-232 port

ACCESS\_DENIED - the port is in use by another program

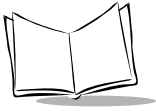
FILE\_NOT\_FOUND - the port does not exist

other - consult WinError.h

### Example

```
if (cspGetCommInfo(COM1) != SERIAL_RS232)
{
    // try another port until an available
    // RS-232 port is found...
```

```
    }  
else  
    {  
        // this is the first available RS-232 port,  
        // display a message...  
    }
```



## Advanced Function Commands

---

Four additional functions provide users with advanced capabilities for reading data as well as setting and retrieving parameters. Knowledge of the CSP protocol is required when dealing with raw data formats.

### *Read Raw Data*

#### Syntax

```
long cspReadRawData(char aBuffer[], long nMaxLength )
```

#### Description

The **cspReadRawData()** function performs an “Upload” command on a CSP device. The raw data read from the CSP device is placed into *aBuffer[]* for up to *nMaxLength* bytes. If the amount of data that is read from the CSP device exceeds *nMaxLength*, the buffer will be truncated at *nMaxLength* bytes. The data in the buffer will be exactly the data received from the CSP device. A knowledge of the CSP protocol and data format is required in order to interpret the device data.

#### Returned Status:

Positive return values indicate the number of bytes actually read from the CSP device.

If the return value was negative:

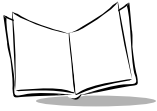
```
COMMUNICATIONS_ERROR  
INVALID_COMMAND_NUMBER  
COMMAND_LRC_ERROR  
RECEIVED_CHARACTER_ERROR  
GENERAL_ERROR
```

#### Example

```
char aBuffer[4096];  
long nBytesRead;  
  
// initialize the COM port for its FIRST use...  
If (cspInit(COM1) == STATUS_OK)  
{  
    // read in the raw data...  
    if ((nBytesRead =  
        cspReadRawData (aBuffer, sizeof(aBuffer))) > 0)
```



```
    {  
    // nBytesRead indicates how many bytes of  
    // data were read  
    }  
else  
    // Error: alert the user to activate the CSP device}
```



## Set Device Parameters

### Syntax

```
long cspSetParam (long nParam, char szString[], long nMaxLength )
```

### Description

The **cspSetParam()** function permits the user to write individual CSP device parameters one at a time as illustrated by the following table.

If this is the last action in a communications session, it should be followed by a power down command.

### Parameters

Where:

*nParam* is one of the following:

TL\_BITS  
VOLUME  
BARCODE\_REDUNDANCY  
USER\_ID  
CONTINUOUS\_SCANNING

*szString[]* contains the new parameter setting

*nMaxLength* determines how many characters in *szString[]* are used in setting the parameters.

<i>NParam</i>	<i>SzString[]</i>	<i>nMaxLength</i>
TL_BITS	char [8]: any eight byte string	8
VOLUME	Array with the first byte set to one of the following values: VOLUME_QUIET VOLUME_LOW VOLUME_MEDIUM VOLUME_HIGH	1

BARCODE_REDUNDANCY	Array with the first byte set to one of the following values: PARAM_OFF PARAM_ON	1
USER_ID	char [8]: any eight byte string	8
CONTINUOUS_SCANNING	Array with the first byte set to one of the following values: PARAM_OFF PARAM_ON	1

## Returned Status

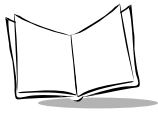
STATUS\_OK  
 COMMUNICATIONS\_ERROR  
 INVALID\_COMMAND\_NUMBER  
 COMMAND\_LRC\_ERROR  
 RECEIVED\_CHARACTER\_ERROR  
 GENERAL\_ERROR

## Example

```

char szString[9];
// create a volume string...
szString[0] = VOLUME_LOW;
szString[1] = 0;
// initialize the COM port for its FIRST use...
If (cspInit(COM1) == STATUS_OK)
{
    // try to change the volume setting...
    if (cspSetParam (VOLUME, szString, 1) == STATUS_OK)
    {
        // The CSP volume setting was changed...
    }
    else
        // Error: alert the user to activate the CSP device
}

```



## Get Device Parameters

### Syntax

```
long cspGetParam (long nParam, char szString[], long nMaxLength )
```

### Description

The **cspGetParam()** function permits the user to read individual device parameters one at a time as illustrated by the following table.

If this is the last action in a communications session, it should be followed by a power down command.

### Parameters

Where:

*nParam* is one of the following:

TL\_BITS  
VOLUME  
BARCODE\_REDUNDANCY  
USER\_ID  
CONTINUOUS\_SCANNING

*SzString[]* will be used to store the retrieved parameter setting

*nMaxLength* determines how many characters from the retrieved parameter are written into *SzString[]*.

nParam	SzString[]	required <i>nMaxLength</i>
TL_BITS	char [8]: any eight byte string	8
VOLUME	Array with the first byte set to one of the following values: VOLUME_QUIET VOLUME_LOW VOLUME_MEDIUM VOLUME_HIGH	1

BARCODE_REDUNDANCY	Array with the first byte set to one of the following values: PARAM_OFF PARAM_ON	1
USER_ID	char [8]: any eight byte string	8
CONTINUOUS_SCANNING	Array with the first byte set to one of the following values: PARAM_OFF PARAM_ON	1

## Returned Status

STATUS\_OK  
 COMMUNICATIONS\_ERROR  
 INVALID\_COMMAND\_NUMBER  
 COMMAND\_LRC\_ERROR  
 RECEIVED\_CHARACTER\_ERROR  
 GENERAL\_ERROR

## Example

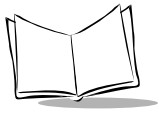
```

char szString[9];

// initialize the COM port for its FIRST use...
if (cspInit(COM1) == STATUS_OK)
{
    // read in the current volume setting
    if (cspGetParam (VOLUME, szString, 1) == STATUS_OK)
    {
        // The CSP volume setting was changed...
        // szString[0] contains the value of
        // the current volume...
        switch ((long) szString[0])
        {
            case VOLUME_QUIET: // Note: long values!
            case VOLUME_LOW:
            case VOLUME_MEDIUM:
            case VOLUME_HIGH:
                break;

            default:

```



## *32-Bit DLL for Windows 95, 98, and Windows NT User Guide*

```
                                // print out error message...
                                break;
                                }
else
    // Error: alert the user to activate the CSP device
}
```

## Interrogate the Device

### Syntax

```
long cspInterrogate ( void )
```

### Description

The **cspInterrogate()** function request a response from the device indicating that it is operating and containing version information. The actual CSP device interrogation reports the communications status, the protocol version, the system status, and the device software version. All of this information can be read back using the appropriate CSP functions.

---

**Note:** *An interrogation is required to establish communications with the device each time the device is activated. If the device goes to sleep, an interrogation is required to re-establish communications. All DLL functions that communicate with the device automatically perform an interrogation. An audible beep indicates that the device is connected before this function returns.*

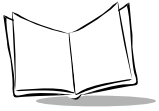
---

### Returned Status

```
STATUS_OK
COMMUNICATIONS_ERROR
INVALID_COMMAND_NUMBER
COMMAND_LRC_ERROR
RECEIVED_CHARACTER_ERROR
GENERAL_ERROR
```

### Example

```
// initialize the COM port for its FIRST use...
if (cspInit(COM1) == STATUS_OK)
{
    if (cspInterrogate() == STATUS_OK)
    {
        // found a CSP device on this port!
    }
}
else
{
    // Error: alert the user to activate the CSP device
```

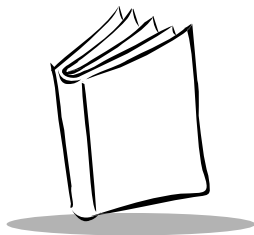


*32-Bit DLL for Windows 95, 98, and Windows NT User Guide*

}

}





## Index

### A

Advanced Function Commands ..... 2-40

### B

Basic Function Commands ..... 2-6  
bullets ..... x

### C

Clear Barcodes ..... 2-8  
Communications Function Commands .... 2-3  
conventions  
    notational ..... x  
CSP Configuration Get Command . s 2-22, 2-29  
CSP Configuration Set Commands ..... 2-22  
CSP Data Get Commands ..... 2-10  
CSP32.DLL  
    Install ..... 1-1  
    redistributing ..... 1-1

### D

Debug Commands ..... 2-37  
DLL Configuration Commands ..... 2-34

### F

Function Definitions ..... 2-1

### G

Get Barcode ..... 2-10  
Get Barcode Redundancy ..... 2-32  
Get Communications Port Information ... 2-38  
Get Continuous Scanning ..... 2-33

Get Device ID ..... 2-12  
Get Device Parameters ..... 2-44  
Get DLL Version ..... 2-36  
Get Protocol ..... 2-14  
Get Retry Count ..... 2-35  
Get Software Version ..... 2-20  
Get System Status ..... 2-16  
Get TL Bits ..... 2-29  
Get User ID ..... 2-18  
Get Volume ..... 2-31

### I

information, service ..... x  
Initialize Communications Port ..... 2-3  
Interrogate the Device ..... 2-47

### M

Miscellaneous Commands ..... 2-36

### N

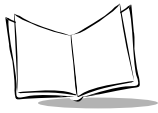
notational conventions ..... x

### P

Power Down ..... 2-9

### R

Read Data ..... 2-6  
Read Raw Data ..... 2-40  
Restore Communications Port ..... 2-5  
Returned Status Definitions ..... 2-1  
    Access\_Denied ..... 2-2



Bad_Param	2-2
Command_LRC_Error	2-2
Communications_Error	2-2
File_Not_Found	2-2
General_Error	2-2
Invalid_Command_Number	2-2
No_Error_Encountered	2-2
Received_Character_Error	2-2
Setup_Error	2-2
Status_OK	2-2

## S

service information	x
Set Barcode Redundancy	2-25
Set Continuous Scanning	2-28
Set Debug Mode	2-37
Set Device Parameters	2-42
Set Retry Count	2-34
Set TL Bits	2-22
Set User ID	2-26
Set Volume	2-24
symbol support center	xi

**part number**  
**Revision Level — Release Date**

